WEST

Generate Collection

L3: Entry 6 of 6 File: USPT Aug 22, 2000

DOCUMENT-IDENTIFIER: US 6108715 A
TITLE: Method and system for invoking remote procedure calls

Detailed Description Text (19):
Having mapped the server thread's stack into the kernel's address space, the kernel continues processing by structuring the server's stack 668 so that it simulates the client stack 641. That is, the kernel sets the server thread's stack pointer and allocates storage for the method's parameters and associated data so that the server stack appears as if the real method 622 was invoked locally. The signatures of the prototype table are used to facilitate this stack construction. The kernel then determines the location of the signature table 663 via the resource table entry's 665 signature table pointer 750. After determining the location of the signature table, the kernel uses the virtual function table index passed in register 648 to determine the appropriate signature table entry 669 for the real method 622.

WEST

Generate Collection | Print

L7: Entry 2 of 6 File: USPT Mar 5, 2002

DOCUMENT-IDENTIFIER: US 6353829 B1
**** See image for Certificate of Correction ****
TITLE: Method and system for memory allocation in a multiprocessing environment

Brief Summary Text (27):
A program executing on a single-threaded processor may have multiple threads that execute concurrently, but not simultaneously. Each of these threads may request that memory be allocated or freed. Conventional memory allocation techniques, however, do not support the concurrent execution of memory allocation or memory free routines. If such routines were executed concurrently, a thread may find the state of the data structures used when allocating and freeing memory to be inconsistent because another thread is in the process of updating the state. Conventional memory allocation techniques may use a conventional locking mechanism (e.g., a semaphore) to prevent the concurrent execution of the memory allocation and memory free routines. Thus, the locked out threads will wait until another thread completes its memory allocation. Such waiting may be acceptable in a single-threaded processor environment, because only one thread can be executing at anytime so the processor may be always kept busy. Such waiting, however, is unacceptable in a multithreaded processor environment because many streams of the processor may be left idle waiting for a thread executing on another stream to complete its memory allocation request.

Other Reference Publication (14):
D.H. Bailey et al., "The NAS Parallel Benchmarks--Summary and Preliminary Results," Numerical Aerodynamic Simulation (NAS) Systems Division, NASA Ames Research Center, California, 1991. p 158-165.

WEST

☐ Generate Collection

L7: Entry 6 of 6 File: USPT Nov 25, 1997

US-PAT-NO: 5692193
DOCUMENT-IDENTIFIER: US 5692193 A

TITLE: Software architecture for control of highly parallel computer systems

DATE-ISSUED: November 25, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Jagannathan; Suresh	Princeton	NJ		
Philbin; James F.	Metuchen	NJ		

US-CL-CURRENT: 709/106; 709/1, 709/315, 711/6

ABSTRACT:

A computer software architecture for controlling a highly parallel computer system comprises several layers of abstraction. The first layer is an abstract physical machine which contains a set of abstract physical processors. This layer may be considered as a microkernel. The next layer includes virtual machines and virtual processors. A virtual machine comprises a virtual address space and a set of virtual processors that are connected in a virtual topology. Virtual machines are mapped onto abstract physical machines with each virtual processor mapped onto an abstract physical processor. The third layer of abstraction defines threads. Threads are lightweight processes that run on virtual processors. In a preferred embodiment the abstract physical machines, abstract physical processors, virtual machines, virtual processors, thread groups, and threads are all first class objects.

21 Claims, 12 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 9

WEST

Generate Collection Print

L5: Entry 7 of 75 File: USPT May 20, 2003

DOCUMENT-IDENTIFIER: US 6567839 B1
TITLE: Thread switch control in a multithreaded processor system

Detailed Description Text (16):
Thread state registers 440 comprise a state register for each thread and, as the name suggests, store the state of the corresponding thread; in this case, a T0 thread state register 442 and a T1 thread state register 444. The number of bits and the allocation of particular bits to describe the state of each thread can be customized for a particular architecture and thread switch priority scheme. An example of the allocation of bits in the thread state registers 442, 444 for a multithreaded processor having two threads is set forth in the table below.

Detailed Description Text (17):
Thread State Register Bit Allocation (0) Instruction/Data 0=Instruction 1=Data (1:2) Miss type sequencer 00=None 01=Translation lookaside buffer miss (check bit 0 for I/D) 10=L1 cache miss 11=L2 cache miss (3) Transition 0=Transition to current state does not result in thread switch 1=Transition to current state results in thread switch (4:7) Reserved (8) 0=Load 1=Store (9:14) Reserved (15:17) Forward progress counter 111=Reset (instruction has completed during this thread) 000=1st execution of this thread w/o instruction complete 001=2nd execution of this thread w/o instruction complete 010=3rd execution of this thread w/o instruction complete 011=4th execution of this thread w/o instruction complete 100=5th execution of this thread w/o instruction complete (18:19) Priority (could be set by software) 00=Medium 01=Low 10=High 11=<Illegal> (20:31) Reserved (32:63) Reserved if 64 bit implementation

Detailed Description Text (39):
To remedy the problem of thrashing wherein each thread is locked in a repetitive cycle of switching threads without any instructions executing, there exists a forward progress count register 420 (FIG. 4) which allows up to a programmable maximum number of thread switches called the forward progress threshold value. After that maximum number of thread switches, an instruction must be completed before switching can occur again. In this way, thrashing is prevented. Forward progress count register 420 may actually be bits 30:31 in the thread switch control register 410 or a software programmable forward progress threshold register for the processor. The forward progress count logic uses bits 15:17 of the thread state registers 442, 444 that indicate the state of the threads and are allocated for the number of thread switches a thread has experienced without an instruction executing. Preferably, then these bits comprise the forward progress counter.

WEST

☐ Generate Collection

L5: Entry 13 of 75 File: USPT Apr 1, 2003

US-PAT-NO: 6542920
DOCUMENT-IDENTIFIER: US 6542920 B1

TITLE: Mechanism for implementing multiple thread pools in a computer system to optimize system performance

DATE-ISSUED: April 1, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Belkin; Ruslan	Mountain View	CA		
Ramachandran; Viswanath	Mountain View	CA		

US-CL-CURRENT: 709/104; 707/10, 707/102, 709/100, 709/101, 709/102, 709/103, 709/105, 709/106, 709/107, 709/108, 709/201 , 709/202, 709/203, 709/226

ABSTRACT:

A mechanism is disclosed for implementing multiple thread pools in a computer system to optimize system performance. In accordance with the invention, a plurality of thread pools is initially allocated within a process space, with each thread pool comprising one or more threads. Each thread pool has a set of characteristics associated therewith, and the characteristics of each thread pool are customized for one or more particular types of service. After the thread pools have been allocated, the system receives one or more requests. When a request is received, it is processed to determine with which thread pool the request is to be associated. This processing is carried out by determining the type of service being requested by the request, and then determining which thread pool is associated with that type of service. Alternatively, this processing is carried out by extracting a set of indication information (e.g. a universal resource identifier) from the request, and then determining which thread pool is associated with that set of indication information. Once the proper thread pool is determined, a thread from that thread pool is used to carry out servicing of the request. Because the request is serviced using a thread from the thread pool customized for the type of service requested, the servicing of the request is optimized. This in turn optimizes system performance.

39 Claims, 6 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 6

WEST

☐ Generate Collection

L3: Entry 3 of 6 File: USPT Nov 27, 2001

US-PAT-NO: 6324492
DOCUMENT-IDENTIFIER: US 6324492 B1

TITLE: Server stress testing using multiple concurrent client simulation

DATE-ISSUED: November 27, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Rowe; Michelle M.	Redmond	WA		

US-CL-CURRENT: 703/13; 370/241, 709/100, 709/203

ABSTRACT:

Method and system for simulating multiple concurrent clients on a network server to stress test the server. Each of one or more processors has one executable software thread, the send data thread, whereby I/O requests are initiated on behalf of all simulated clients. Each of the one or more processors has another executable software thread, the receive data thread, whereby I/O response data is received from the server on behalf of all simulated clients. A software object, such as a completion port, that is capable of detecting completion of an executable event allows the stress test to function using only two executable threads per processor. The efficiency of the server can be measured as the send data thread initiates I/O requests and as the server responds thereto. The stress test is flexible in that the number of simulated clients and the I/O requests initiated can vary widely. Moreover, many different types of network servers can be tested.

26 Claims, 12 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 12

WEST

Generate Collection

L3: Entry 2 of 6 File: USPT Jun 11, 2002

DOCUMENT-IDENTIFIER: US 6405326 B1
TITLE: Timing related bug detector method for detecting data races

Brief Summary Text (21):
In case the program contains at one of its points N parallel structures, then it can be split into at most N parallel threads. Therefore if a multi-processors computer is available to execute this program, then the OS can allocate each of the parallel threads to a different processor. Alternatively, in the case of single processor architecture, the OS can simulate the allocation of threads to respective processors.

WEST

Generate Collection

L7: Entry 4 of 6 File: USPT Mar 20, 2001

DOCUMENT-IDENTIFIER: US 6205414 B1
TITLE: Methodology for emulation of multi-threaded processes in a single-threaded operating system

Detailed Description Text (14):
With reference now to FIG. 3A-3B, timing diagrams showing emulation of multi-threaded processing during execution of master and slave code threads in accordance with a preferred embodiment of the present invention are depicted. FIG. 3A shows processing alternating between a single master code thread and a single slave code thread. The duty cycle between the number of clocks allocated to the master code thread and the number of clocks allocated to the slave code thread is controlled by the time-out period of the timer and the maximum latency enforced on the slave code thread. A few clock cycles may be consumed each time processing is switched between master and slave code threads by the interrupt service routine, which must save the state of the master code thread processing, if necessary, reset the timer, etc. FIG. 3B depicts processing alternating between a single master code thread and a slave code thread component selected, in round-robin fashion, from a group of n slave code thread components.

Detailed Description Text (18):
Utilizing the methodology of the present invention allows separate and disparate code bodies to be tied together and executed concurrently. The underlying, single-threaded operating system remains essentially intact. The workload and expense for merging single-threaded functionality into a simulated multi-threaded environment is thus substantially reduced.

WEST

Freeform Search

Database: US Patents Full-Text Database / US Pre-Grant Publication Full-Text Database / JPO Abstracts Database / EPO Abstracts Database / Derwent World Patents Index / IBM Technical Disclosure Bulletins

Term:

Display: 10 Documents in Display Format: TI Starting with Number 1

Generate: ○ Hit List [✓ Hit Count] ○ Side by Side ○ Image

Search Clear Help Logout Interrupt

Main Menu | Show S Numbers | Edit S Numbers | Preferences | Cases

Search History

DATE: Thursday, November 13, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT; PLUR=YES; OP=ADJ		
L8	thread$ near5 allocat$ near5 static$	3	L8
L7	L5 and simulat$	6	L7
L6	L5 same simulat$	0	L6
L5	L4 same l2	75	L5
L4	(state or status) near3 thread	1680	L4
L3	L2 same simulat$	6	L3
L2	thread$ near5 allocat$	722	L2
L1	6430619	2	L1

END OF SEARCH HISTORY